CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2020
Doc 3 Python
Jan 28, 2020

Copyright ©, All rights reserved. 2020 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/openpub/) license defines the copyright on this document.

Python

Interpreted dynamically type language

Created by Guido van Rossum

Version 1.0 1991

Version 2.0 2000
    1/2 == 0

Version 3.0 2008
    1/2 == 0.5
    1//2 == 0

Popular IDEs
    PyCharm
    Visual Code

```
width = "cat"
width = 4
height = 5
area = width * height
```

Tutorial
    https://docs.python.org/3/tutorial/index.html

Documentation
    https://docs.python.org/3/index.html

Why Popular

Simple & expressive syntax

Interactive
- Faster development

Large number of libraries

Plays well with
- C
- Fortan

Problems with Python

Slow

Python 2 verse Python 3

GIL - Global Interpreter Lock

Strings

```
a = 'A String'
b = " Another string"

a[0] == 'A'
a[0:3] == 'A S'
a[-1] == 'g'

c = a + b

d = "Cat " "in " "the hat"
```

```
raw_string = r"Special char \t are treated as normal chars"
```

If, for

```
x = int(input("Please enter an integer: "))

if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

Boolean operators

and, or, not

```
words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w))
```

```
# Loop over a slice copy of the entire list.
for w in words[:]:
    if len(w) > 6:
        words.insert(0, w)
```

Warning

```
words = ['cat', 'window', 'defenestrate']
for w in words:
print(w, len(w))
```

Interpreter error

Indentation is required

==, is

a == b

Compare values

a is b

Point to same object

Functions

```
def fib(n):  # return Fibonacci series up to n
    """Return a list containing the Fibonacci series up to n."""
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Default Values

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

```
parrot(1000)                                          # 1 positional argument
parrot(voltage=1000)                                  # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM')             # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000)             # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump')         # 3 positional arguments
parrot('a thousand', state='pushing up the daisies')  # 1 positional, 1 keyword
```

Python

```
def ask_ok(prompt, retries=4, reminder='Please try again!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
        print(reminder)
```

lambda

```
def make_incrementor(n):
    return lambda x: x + n

f = make_incrementor(42)
g = make_incrementor(10)
f(0)                                # 42
g(2)                                # 12
f(1)                                # 43
```

Python lambda's can only contain one expression

Nested Functions

```
def running_sum(n):
    sum = n
    def perform_sum(x):
        nonlocal sum
        sum += x
        return sum
    return perform_sum

f = running_sum(0)
f(10)
f(2)
```

Use 4-space indentation, and no tabs.

Wrap lines so that they don’t exceed 79 characters.

Use blank lines to separate
    functions and classes,
    larger blocks of code inside functions

When possible, put comments on a line of their own

Use docstrings

Use spaces around operators and after commas
    a = f(1, 2) + g(3, 4)

Naming convention
    CamelCase for classes and
    lower_case_with_underscores for functions and methods.
    Always use self as the name for the first method argument

Arrays, Lists, Tuples - Sequence Types

array
- Wrapper for C-array
- Homogeneous values
- Growable, Mutable
- Consider using NumPy instead

List
- Heterogeneous values
- Growable, Mutable

```
x = [1,2, "cat"]
x.append(1.2)
x[0] = 9
```

Tuple
- Heterogeneous values
- Immutable

```
x = (1,2, "cat")
```

List Methods

list.append(x)
list.extend(iterable)
list.insert(i, x)
list.remove(x)
list.pop([i])
list.clear()
list.index(x[, start[, end]])
list.count(x)
list.sort(key=None, reverse=False)
list.reverse()
list.copy()

List Compressions

```
squares = [x**2 for x in range(10)]
```

```
[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
[(x, y)
 for x in [1,2,3]
 for y in [3,1,4]
 if x != y]
```

```
from math import pi
[str(round(pi, i)) for i in range(1, 6)]
```

```
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

Sets

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

'orange' in basket              # True


a = set('abracadabra')
b = set('alacazam')
a - b                                   # {'r', 'd', 'b'}
```

Dictionaries

```
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
```

```
dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```

```
{x: x**2 for x in (2, 4, 6)}                    #  {2: 4, 4: 16, 6: 36}
```

Formating

```
table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
for name, phone in table.items():
    print(f'{name:10} ==> {phone:10d}')
```

```
Sjoerd     ==>       4127
Jack       ==>       4098
Dcab       ==>       7678
```

File IO

```
with open('workfile') as f:
    read_data = f.read()
```

File closed when block ends

```
for line in f:
   print(line, end='')
```

```
f.readlines()
```

Exceptions

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")


raise NameError('HiThere')
```

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
```

Classes

```
class Dog:
    kind = 'canine'         # class variable shared by all instances

    def __init__(self, name):
        self.name = name    # instance variable unique to each instance

     def __str__(self):
        return self.name
```

```
d = Dog('Fido')
d.name
d.name = "Rover"
d.size = "Large"
d.size


def speak():
    return "Bark"

d.speak = speak
d.speak()
```